POLYNOMIAL ALGORITHM FOR SINGLE MACHINE SCHEDULING WITH TWO PROCESSING TIMES TO MINIMIZE MEAN TRADINESS AND EXTENSIONS

A Thesis Submitted
in Partial Fulfilment of the Requirements
for the Degree of

MASTER OF TECHNOLOGY

by

G. RAMAKRISHNA MOHAN

to the

INDUSTRIAL AND MANAGEMENT ENGINEERING PROGRAMME
INDIAN INSTITUTE OF TECHNOLOGY KANPUR
MARCH, 1986

CERTIFICATE

This is to certify that the present work on "Polynomial Algorithm for Single Machine Scheduling with two Processing Times to Minimize Mean Tardiness, and Extensions," by G. Ramakrishna Mohan has been carried out under our supervision and has not been submitted elsewhere for the award of a degree.

(S. Sadagopan)
Assistant Professor
Industrial and Management
Engineering Programme
Indian Institute of Technology
Kanpur 208 016 (UP)

(Ashok K. Mittal)
Professor
Industrial and Management
Engineering Programme
Indian Institute of Technology
Kanpur 208 016 (UP)

March, 1986

Acknowledgements

I wish to express my deep sense of gratitude to my thesis supervisors, Dr. A.K. Mittal and Dr. S. Sadagopan, for their expert guidance, constant encouragement, and invaluable advice throughout the course of my thesis work.

I thank Mr. R.B. Yarlagadda, Mr. Prasad, Mr. Sattanathan, Mr. Gopinath and other friends for their help.

I am grateful to Mr. R.N. Srivastava and Mr. B.R. Kandiyal for their neat typing and cyclostyling work.

- G. Rama Krishna Mohan

Abstract

In this thesis, single machine scheduling problem with minimum total tardiness (mean tardiness) as objective is considered for the special case where the jobs have processing times limited to two values only. A polynomial time algorithm is developed for this case. Using the ideas developed for this algorithm, two heuristic procedures have been developed for the general single machine scheduling problem with minimum total tardiness as the objective. Computational performance of the heuristics have been compared with the performance of the Wilkerson-Irwin heuristic. In general, the heuristicsdeveloped in this thesis give improved solutions though these heuristics take more time.

Contents

Chapter I

INTRODUCTION

Introduction

Scheduling can be defined as "the allocation of resources over time to perform a collection of tasks" [1]. The term facilities is often used instead of resources and the tasks to be performed may involve a variety of different operations. Sequencing defines the process of arranging the order in which the tasks are to be performed.

Scheduling is a particularly important function in the field of production and operations management and much of the terminology used is derived from this source. The resources to be allocated may be referred to as machines, processors or facilities. The tasks are defined as operations that have to be performed on jobs by the machines. The term job is commonly used in this context to designate a single item, or an entire batch of items, that require processing on the machines. Technological and other prescribed constraints on the order in which jobs may be processed are important and accordingly scheduling problems can be classified as:

General job-shop Scheduling

- Where every job may have a different routing through the machines.

Flow-shop Scheduling

- Where every job has the same routing through the machines.

Permutation Scheduling

- Where the same job sequence applies on all the machines.

The second and third problem classes are really special cases of the general job-shop scheduling problem.

There are many problems of scheduling outside the traditional production field such as arranging appointments for patients in hospital, processing of programs by computer and loading containers. In many cases, although the terminology describing the resources to be allocated, and tasks to be performed may be different, the problems have equivalents in the production context. So the same scheduling theory has wider applicability.

The production scheduling problems are classified based on the following five dimensions:

1. Requirement Generation
2. Processing Complexity
3. Scheduling Criteria
4. Nature of Requirement Specification
5. Scheduling Environment,

The first dimension, requirements generation, is a key distinction. Requirements may be generated either directly by customers orders or indirectly by inventory replenishment dimensions. The distinction is made in terms of an open shop versus closed shop. In an open shop all production orders are by customer request and no inventory is stocked in a closed shop all customer requests are serviced from

inventory and production tasks are generally a result of inventory replenishment decisions. The production scheduling problem is quite different depending on the requirements generation. For open shop, the production scheduling in its simplest form is a sequencing problem in which open orders are sequenced at each facility. For the closed shop, production scheduling involves not only sequencing decisions but also lot sizing decisions associated with inventory replenishment process. Although a pure open or pure closed shop is rare, we assume that the decision whether a problem is open shop or closed shop has been made.

The second dimension, processing complexity, is concerned primarily with the number of processing steps associated with each production task or item. A common breakdown of this dimension are

- One stage, one processor (facility)
- One stage, parallel processors
- Multistage flowshop
- Multistage jobshop.

In the one stage, one processor problem, which is also termed as single machine problem, all tasks require one processing step which must be done on the one production facility.

The one-stage, parallel processor problem is similar to the single machine problem except that each task requires a single processing step which may be processed on any of the parallel processors. For the multi-stage problem,

each task requires a set of distinct facilities, where typically there is a strict precedence ordering of the processing steps for a particular task. The flowshop problem assumes that all tasks are to be processed on the same set of facilities with an identical precedence ordering of the processing steps. The jobshop problem is the most general production scheduling problem in this classification; here there are no restrictions on the processing steps for a task and alternative routings for a task may be allowed. The above characterization of processing complexity seems capable of capturing the nature of most production environments.

The third dimension, scheduling criteria, indicates the measures which can be used to evaluate the schedules. Two broad classes of criteria are schedule-cost and schedule-performance. The cost associated with a particular schedule includes the fixed costs, variable production and overtime costs, inventory holding costs, shortage costs for meeting downtime etc. The different scheduling performance measures are utilisation level of the production resources, percentage of late tasks, the average or maximum tardiness for a set of tasks etc.

In most production environments, the schedule evaluation is based on a mixture of both cost and performance criteria. In the literature, openshop problems are evaluated with schedule performance criteria and closed shop with minimum cost function.

The fourth dimension is the nature of the requirement specification. The problem is classified as deterministic

problem if all the data involved are deterministic (processing times, sequences, technological constraints, availability of facility etc.); otherwise it is stochastic.

The fifth dimension is the scheduling environment. The problem is static if none of the initial data changes over-time (e.g. when all the jobs that are to be considered) are available simultaneously at the beginning of the scheduling period. The problem is dynamic if the data is subject to change with time e.g. when the jobs arrive intermittently during the scheduling period.

While an actual real life scheduling problem is likely to have a complex structure, it is easier to first analyse some problems with simple and well defines structure. The ideas generated from these problems can be used to understand and analyse more complex problems.

In a production scheduling problem, the simplest case is that of an open-shop, one machine, deterministic and static environment.

1.2 Single Machine Sequencing Problem:

The pure sequencing problem is a specialised scheduling problem in which an ordering of the jobs completely determines a schedule. The simplest pure sequencing problem is the one in which there is only one machine.

The basic single machine problem is characterised by the following conditions, Baker [1].

1. A set of n independent, single-operation jobs is available for processing at time zero.

2. Set up times for the jobs are independent of job sequence and can be included in processing times.
3. Job descriptors are known in advance.
4. One machine is continuously available and is never kept idle while work is waiting.
5. Once processing begins on a job, it is processed to completion without interruption.

The basic information required to describe the jobs in the deterministic single machine case are

Processing time t_j:- The amount of processing required by job j.

Ready time r_j:- The point in time at which job j is available for processing.

Duedate d_j:- The point in time at which the processing of job j is due to be completed.

The processing time t_j will generally include both direct processing time and facility set-up time. The ready time r_j can be thought of as an arrival time - the time when job j appears at the processing facility. We shall assume that all jobs are available at the beginning itself.

The following statistics can be computed from the schedules.

Completion time C_j: The time at which job j is finished.

Flow time F_j: The amount of time job j spends in the system; $F_j = C_j - r_j$.

Lateness L_j: The amount of time by which the completion time job j exceeds its due date $L_j = C_j - d_j$.

Tardiness T_j: The lateness of job j if it fails to meet its due date; $T_j = \max(0, L_j)$.

Schedules are generally evaluated by aggregating quantities that involve information about all jobs, resulting in one-dimensional performance measures.

Different performance measures are

Mean flow time : $\bar{F} = \frac{1}{n} \sum_{j=1}^{n} F_j$

Mean tardiness : $\bar{T} = \frac{1}{n} \sum_{j=1}^{n} T_j$

Max flowtime : $F_{max} = \max_{1 \leq j \leq n} \{F_j\}$

Max tardiness : $T_{max} = \max_{1 \leq j \leq n} \{T_j\}$

Number of Tardy jobs: $N_T = \sum_{j=1}^{n} \delta T_j$

where $\delta(x) = 1$ if $x > 0$

$= 0$ otherwise.

For most of the performance measures for single machine problem, good i.e., polynomial time algorithms are available to obtain the optimal solutions. However for the weighted tardiness, no such algorithm is available. While the weighted tardiness problem has been shown to be a difficult

problem i.e. NP-complete [9], it is possible to obtain polynomial time algorithms for some special cases. These specially structured problems can be used to approximate the actual problem and to obtain approximate solutions to more complex problems.

In this thesis, we shall consider one such simple structure. The problem under consideration is the unweighted mean tardiness problem in a single machine, deterministic, static and open shop environment with further restriction that the processing times are limited to only two distinct numbers. Essentially the environment indicates a situation where all the jobs can be classified only in two groups for the purpose of processing time or where the shop takes only two types of jobs, requiring distinct processing times. The ideas developed for this problem are further used to develop two heuristics for the case of single machine, scheduling problem with arbitrary processing time and due date for mean tardiness measure. Further attempt is made to identify the problem structure over which a particular heuristic will work better. This approach may lead to construction of a decision support system in which first the problem will be analysed for its structure with respect to certain specified parameters and then a suitable heuristic will be selected from the heuristic bank.

1.3 Outline of the Thesis:

After a brief introduction in Chapter I, the survey of relevant literature is outlined in Chapter II. A polynomial

algorithm for a restricted class of single machine problem is developed in Chapter III. In this algorithm, the processing times of all jobs is restricted to only two distinct values. An extension of the ideas developed in this algorithm is extended to general case in the form of two heuristic procedures. Chapter IV reports some computational studies where the computational performance of the algorithm and the heuristics are reported and analysed. An attempt is made to identify the problem structures (processing time and due dates) for which these heuristics will perform well. Chapter V outlines the conclusions and some suggestions for future work.

Chapter II

LITERATURE REVIEW

The single machine, n-job sequence independent total tardiness problem was first considered by McNaughton [3]. To get an optimal solution, a complete enumeration procedure was given.

Held, Karp and Lawler [1] presented a dynamic programming formulation which require 2^n possible subsets of n jobs. Let J denote the set of jobs which has not been scheduled at any stage and J' be its complement. Let q_J be the completion of all jobs that are sequenced. Then the principle of optimality states, that no matter how the jobs in J' are sequenced, the jobs in J must be sequenced optimally subject to the constraint that no job may begin prior to q'_J. Dynamic programming is typical of many general purpose procedures of combinatorial optimization. The effort required to solve the problem grows at an exponential rate with the increasing problem size. However this approach is more efficient than complete enumeration of all feasible solutions.

Emmons [2] proved results which establish the relative order in which pairs of jobs are processed in an optimal sequence. It was claimed that, frequently these dominance results permit the jobs to be completely ordered, thus solving the problem without any searching. He also established general conditions under which sequencing in order of non-decreasing processing times and sequencing in order of non-decreasing due

dates are optimal. These results permit us to eliminate many feasible schedules from consideration while searching the solution space. Emmons also proposed a branch and bound algorithm which employs the above results.

The hybrid algorithm devised by Srinivasan, presented by Baker, [1] improves on dynamic programming by exploiting some specialised dominance properties that can be developed for the problem. Here in the first phase of the algorithm, he used two dominance properties that have been developed by Emmons.

With these properties most of the jobs can be sequenced and only a subset of the jobs remains to be sequenced. So in the 2^{nd} phase of the algorithm, the reduced problem is solved by dynamic programming. In this method, four dominance properties are used, by which the computational effort required to solve the reduced problem is considerably reduced. This algorithm is a curtailed enumeration procedure because enumeration of all sequences is curtailed by the characteristics found in the data.

For the case of linear penalty functions, Lawler [5] has given a linear programming formulation which require $n+2T$ constraints where T is total processing time for all n jobs. A simplex algorithm with modification to handle a particular size of restricted entry constraint is developed to solve this formulation. It is also possible to formulate this problem as mixed integer linear programming requiring n^2 constraints, 3n continuous variables and $n(n-1)/2$ zero variables.

number of jobs

Elmaghraby [4] presented a network model which is equivalent to the earlier dynamic programming formulation.

Shwimer [5] has considered the problem with the objective function which is to minimise the sum of incurred tardiness penalties. Elmaghraby shortest route formulation is the underlying structure for the branch and bound algorithm, designed to solve this problem.

Rinnooy Khan & others [6] have considered a generalised cost function for the tardiness penalties. They have developed a branch and bound method to solve this problem. In this algorithm some nodes are eliminated from consideration using results that have been proved for non-decreasing cost functions and a good lower bound is developed by which enumeration is curtailed considerably. The computational performance of this algorithm for 15-20 job problem is shown to be good.

Later Fisher [8] presented a branch and bound algorithm to minimise total tardiness. The algorithm uses dual problem to obtain a good feasible solution and an extremely well sharp lower bound on the optimal objective value. To derive the dual problem, single machine is considered as an imposing constraint for each time period. A dual variable is associated with each of these constraints and a Lagrangian problem is constructed in which the dualised constraints appear in objective function. A lower bound is obtained by solving the Lagrangian problem with fixed multiplier values. The major theoretical results of the paper in an algorithm which solves the Lagrangian problem in $O(n^2p)$ where n is the number of jobs and p is the average processing time. The

search of multiplier values which maximise the lower bound leads to the formulation and optimisation of the dual problem. The bounds obtained are so sharp that very little enumeration or computer time is required to solve even large problems.

When the weights of the jobs are agreeable (i.e., $p_i < p_j$ implies $W_i \geq W_j$), Lawler [11] has developed a pseudo-polynomial algorithm. This approach does solve the well-known mean tardiness problem (all $W_i = 1$). However the computational complexity of that problem is still open, which means that the problem has not been shown to be either NP-hard or polynomially solvable.

Even though the above algorithms lead to an optimal solution, the time required to solve a reasonable size problem is prohibitive. So it is reasonable to consider suboptimal methods or heuristic procedures which are capable of obtaining good solutions very quickly but do not guarantee optimality. The heuristics that are available are Wilkerson-Irwin algorithm, Neighbourhood search techniques and Beam search heuristic.

The Wilkerson-Irwin [1] heuristic uses a decision rule that employs job comparisons in the construction of a sequence. The algorithm involves two ordered lists: a list of unscheduled jobs and a list of scheduled jobs. The scheduled list is a partially completed job sequence that is subject to possible revisions and the unscheduled list contains the remaining jobs in EDD order. At each stage, the first job on the unscheduled list is removed in order to implement the decision rule; this is called pivot job.

Let α be index of last job on the scheduled list

β be index of the pivot job

γ be index of first job on unscheduled list

F_α be the completion of α.

At each stage, the algorithm applies pairwise decision rule to jobs β and γ where $d_\beta \leq d_\gamma$. If $F_\alpha + \max(t_\beta, t_\gamma) \leq d_\gamma$ or if $t_\beta \leq t_\gamma$ then β preceeds γ. If it fails, then rule is applied to jobs α and γ. If $F_\alpha + \max(t_\alpha, t_\gamma) \leq \max(d_\alpha, d_\gamma)$ or if $t_\alpha \leq t_\gamma$ then γ follows α and γ is added to scheduled list. If this decision rule fails, a jump condition results. Under this, α is removed from the scheduled list and placed on the unscheduled list in EDD order. The decision rule is then applied to job γ and the job that preceeded α on the scheduled list. The jump condition occurs infrequently, but it may be employed in succession in order to schedule job γ. This heuristic performs well computationally. On an average the heuristic solution values were about 2% higher than optimum values.

In neighbourhood search techniques, a sequence that is going to be an initial seed is obtained and it is evaluated with respect to a given measure of performance. For generating the initial seed, some rule like SPT is followed. Then all the sequences that are in the neighbourhood are generated and any generating mechanism like adjacent pairwise can be used. These sequences are evaluated. If the performance measures has not improved compared to the initial seed, it is stopped. Otherwise one of the improved sequences in the

neighbourhood is taken and the process is repeated. This method has given reasonable good solution for the mean tardiness problem.

Siow and Morton [7] present another heuristic method called Beam search heuristic. This technique, originally developed for speech recognition can be used for exploring search tree to find a good schedule. In this procedure, each of the newly created node is evaluated first using an evaluation function. A subset of these that are considered promising based on the earliar evaluation is retained and the rest are ignored. The nodes retained from the beam and the maximum number that may be retained at any level of the tree whenever a selection has to be made is known as "beam width". Siow & Morton also investigated the ways in which a given heuristic such as priority rule may be used to generate information to guide the search. Their experiments demonstrate the potential of the Beam search strategy as a very efficient search method and the importance of considering how heuristics may be used to obtain maximum "mileage" from a search technique and limited knowledge about a problem.

Chapter III

A POLYNOMIAL ALGORITHM AND HEURISTICS

3.1 Introduction:

In this chapter, we develop a polynomial time algorithm to solve the single machine scheduling problem with arbitrary due dates when the processing times of the jobs are restricted to only two values a and b and the objective is to minimise mean tardiness. The ideas developed for this problem are used to develop two heuristics for the problem with arbitrary times.

Notation

- a, b - the two distinct values of the processing times ($a < b$)
- p_j - processing time of job, j, $p_j \in \{a\ b\}$
- d_j - due date of job j
- t_j - tardiness of job j
- e_j - earliness of job j
- $i \leftarrow j$ - job i precedes job j.

In the next section we develop a set of decision rules under several sets of conditions. These rules are used in the algorithm to move from an arbitrary sequence to an optimal sequence.

3.2 Decision Rules:

3.21 Consider the following situation

Situation I

Job index		i	j	
Processing time		a	b	
Due date		d_i	d_j	

S = Starting time of job i.

With respect to situation I, Rules 1 to 2 are derived in Lemma 1 to 2 respectively. In the proofs we have used T to indicate the sum of tardiness of jobs i and j in the sequence mentioned above and T' to indicate the sum of tardiness of jobs i and j after interchanging the positions of job i and j in the sequence.

Case	Due date relationship	Earliness of job i	Tardiness of job i	Earliness of job j	Tardiness of job j	Decision rule
1	$d_i \leq d_j$	-	-			i ← j in optimal sequence
2	$d_i > d_j$	$e_i > (b-a)$	-	-	$t_j > 0$	i can be moved to right

Table 3.1 Decision Rules 1-2.

Lemma 1: In situation I if $d_i \leq d_j$, then in an optimal sequence i ← j [5]

Lemma 2: In situation I, if $d_i > d_j$ then if $e_i > (b-a)$ and job j is tardy then i can be moved to right (i.e., i ≠ j).

Proof: $T = t_j$; $T' = \max(t_j - a, 0) + \max(b - e_i, 0)$

Case (i): $t_j \leq a$

$$T = t_j;\ T' = b - e_i < a$$

since $e_i > b-a$

Since $e_i > b-a$; $d_i > S+b$ $\quad d_i = S+b+K$ (say)

$d_j < S+b+K$ (since $d_i > d_j$)

$e_i = b-a+K$ and $t_j > a-K$

$\therefore\ T' = a-K < T.$

Case (ii): $t_j > a$;

$$T' = (t_j - a) + (b - e_i)$$

$$= t_j + (b-a - e_i) < t_j$$

since $e_i > b-a$

$T' < T.$

Therefore the tardiness of schedule can be decreased by moving i to the right (i.e., $i \neq j$). Hence the proof.

3.22 Consider the following situation.

Situation II

Job index		i	j	
Processing time		b	a	
Due date		d_i	d_j	

$d_i \quad d_j$

With respect to Situation II, Rules 3 to 4 are derived in Lemma 3 to 4 respectively.

Case	Due date relationship	Earliness of job i	Tardiness of job i	Earliness of job j	Tardiness of job j	Decision rule
3	$d_i < d_j$	-	$t_i > 0$	-	$t_j > a$	job j can be moved to the left
4	$d_i < d_j$	$e_i > 0$	-	-	$t_j > a$	such a case will not arise

Table 3.2 Decision Rules 3-4.

Lemma 3: In situation II,

if tardiness of job j, $t_j > a$ and job i is tardy

then job j can be moved to the left (i.e., $i \neq j$)

Proof: $T = t_i + t_j$

$$T' = (t_i + a) + \max(t_j - b, 0)$$

Case (i): $a < t_j \leq b$; $T' = t_i + a < T$

Case (ii): $t_j > b$; $T' = t_i + a + t_j - b = t_i + t_j + (a-b) < T$

Therefore tardiness of the schedule can be decreased by moving job j to the left (i.e., $i \neq j$).

Lemma 4: In situation II

job j is tardy, $t_j > a$ and job i is early,

such a case will not arise because of due date relationship.

$d_i > S+b$; $d_i = S+b+K$ (say) if job i is to be early.

$d_j < S+b$ if job j is to be tardy and $t_j > a$.

$\Rightarrow$ $d_i > d_j$ which contradicts the due date relationship.

Consider a sequence in which

$$p_{[i]} = p_{[i+1]} = b;\quad p_{[i+2]} = \cdots = p_{[i+K+1]} = a$$

$$d_{[i]} \leq d_{[i+1]} \leq d_{[i+2]}$$

b	b	a	a	. . .	a
i	i+1	i+2	i+3		i+K+1

Let T_B denote starting time of job $i = \sum_{l=1}^{i-1} p_{[l]}$

Let S be $\{[i+2] \ldots [i+K+1]\}$

B be $\{1 \ldots [i-1]\}$

D be $\{[i+K+2] \ldots n\}$.

Let sequence P be B [i] [i+1] S D

sequence Q be B [i] S [i+1] D

sequence R be B S [i] [i+1] D.

Lemma 5: If the tardiness of sequence P < Tardiness of sequence Q

then tardiness of sequence P < Tardiness of sequence R.

Proof:

If all the jobs with processing time a have tardiness $\leq b$ in sequence P, then in sequence Q, the tardiness of these jobs will be zero. Hence in sequence R, there will not be any decrease in tardiness since the tardiness of jobs in S will not decrease and tardiness of job [i] may increase. Hence in this case, tardiness of sequence P < tardiness of sequence Q.

Now, we consider a situation when there is atleast one job in S whose tardiness is $> b$ in sequence P.

Let earliness of job $[i+1]$ in sequence P be x. Since we did not interchange $[i+1]$ and $[i+2]$, the earliness of job $[i+1]$ in sequence P is $< a$. For any two consecutive pair of jobs (m, n) in S, we observe that

Tardiness of n $\leq$ (Tardiness of job m) + a.

as all jobs with processing time a are in EDD.

Therefore tardiness of job $[i+2] \leq a - x$

$$[i+3] \leq 2a - x$$

$$\vdots$$

$$[i+\overline{s+1}] \leq sa - x$$

$$[i+\overline{s+2}] \leq (\overline{s+1}\ a-x)$$

where $[i+\overline{s+2}]$ is the first job encountered in sequence P as we move right from job $[i+2]$, whose tardiness is $> b$.

$b <$ Tardiness of job $[i+\overline{s+2}] \leq b + a$.

Let T_P, T_Q, T_R denote the tardiness of sequences P, Q and R respectively also t^X_i denote the tardiness of the job $[\,]$ in sequence X.

Given condition $T_Q > T_P$.

$$\therefore\ T_Q = T_P + (Ka-x) - [(a-x) + (2a-x) + \ldots + (sa-x) + \underbrace{b + b + \ldots + b}_{(K-s-1)\text{ jobs.}}]$$

$$T_Q - T_P > 0$$

Hence $(Ka-x) > [(a-x) + (2a-x) + \ldots + (sa-x)] + (K-s-1)b$.

$$T_R - T_Q = Ka - x - b - \left[0 + 0 + \ldots + 0 + a + \{\min(t^Q_j, b) \text{ of the remaining } (K-s-1) \text{ jobs}\}\right]$$

$$> \left[(a-x) + (2a-x) + \ldots + (sa-x) + (K-s-2)b\right] - \left[0 + 0 + \ldots + 0 + a + \{\min(t^Q_j, b) \text{ of the remaining } (K-s-1) \text{ jobs}\}\right]$$

As $t^Q_{i+s+2} < a$ and $\min(t^Q_j, b) < b$

$$i+s+3 \leq j \leq i+K+1$$

$$T_R - T_Q > (a-x) + (2a-x) + \ldots + (sa-x) + (K-s-1)b - \left[a + (K-s-1)b\right].$$

$$T_R - T_Q > (a-x) + (2a-x) + \ldots + (sa-x) - a$$

As $x < a$

$$T_R - T_Q > 0 + a + \ldots + (s-1)a - a$$

$$> 0$$

Hence $T_R > T_Q$.

Hence the proof.

Let S_p denote the sequence

$$b_m\ b_{m-1} \ldots b_p\ a_1\ a_2 \ldots a_K\ b_{p-1} \ldots b_1$$

where $b_m = b_{m-1} = \ldots = b$

$a_1 = a_2 = \ldots = a$

and $T(S_p)$ denote tardiness of sequence S_p.

<u>Theorem</u>

If $T(S_{j-1}) > T(S_{j-2})$

then $T(S_j) > T(S_{j-1})$ for $j \geq 3$.

Proof:

For $j = 3$, the above theorem is proved in Lemma 5.

Consider the case where $j = p > 3$.

Then the sequences S_{p-2}, S_{p-1}, S_p will be

$b_m, b_{m-1}, \ldots, b_p, b_{p-1}, b_{p-2}, a_1, a_2, \ldots, a_K, b_{p-3} \ldots$

$b_m, b_{m-1}, \ldots, b_p, b_{p-1}, a_1, a_2, \ldots, a_K, b_{p-2}, b_{p-3} \ldots$

$b_m, b_{m-1}, \ldots, b_p, a_1, a_2, \ldots, a_K, b_{p-1}, b_{p-2}, b_{p-3} \ldots$

respectively.

The elements before b_p and after b_{p-3} (including this) are same in all the three sequences.

Then these three sequences S_{p-2}, S_{p-1}, S_p are like the sequences P, Q, R in Lemma 5.

Hence if $T(S_{p-1}) > T(S_{p-2})$

then $T(S_p) > T(S_{p-1})$

Hence the proof.

3.3 Algorithm:

Based on the decision rules outlined earlier, we develop an algorithm for two-processing time problem.

Input: Number of jobs = n

p_i (1, 2 ... n) denote processing time of each job $i \in \{a, b\}$.

d_i (1, 2 ... n) due date of each job i.

Let K denote the number of jobs with processing time of a units.

Step 1: Arrange all the jobs with processing time a in EDD order in the first K positions of the sequence. The rest of the positions are filled by remaining jobs with processing time b in EDD order.

Step 2: Construct a set of n intervals as follows:

$$[0\ b]\ [b+1\ \ b+a]\ \ldots\ [b+\overline{K-2}\ a+1\ \ b+\overline{K-1}\ a]$$
$$[b+\overline{K-1}\ a+1\ \ 2b+\overline{K-1}\ a]\ \ldots\ [\overline{N-K}\ b+\overline{K-1}\ a+1$$
$$\overline{N-K}\ b+Ka]\ .$$

Index the intervals as 1, 2, 3 ... n respectively.

Step 2.1: Construct a sequence by assigning the jobs whose processing time = a using the following rules.

(i) Select a job j, whose due date is lowest among those still unassigned to any position.

(ii) Locate an interval '1' such that d_j lies in that interval.

(iii) If 1^{th} position in the sequence is free then assign the job j to position 1. Otherwise see if the n^{th} position is free or not. If n^{th} position is not free, go to Step (iv)(a). Otherwise assign job j to the first unoccupied position encountered as we move towards right from 1^{th} position.

(iv) If the job j is such that d_j lies in the last interval and if last position is already occupied then

a) Find the first unoccupied position encountered as we move left from last position towards the first position. Let it be m^{th} position.

b) Move all jobs assigned to $(m+1)^{st}$ to n^{th} positions by one position to the left.

c) Assign job j to the last position.

Repeat Step 2.1 until all the jobs with processing time = a are assigned.

Step 2.2: Fill up the remaining positions in the sequence with the jobs having processing time = b; maintaining the EDD order.

Step 3: Let A and B be two jobs such that $p_A = a$ and $p_B = b$ then if job B is preceding job A and $d_B \geq d_A$ then interchange the positions of job A and job B.

The Step 3 is repeated until no such interchanges are possible anywhere in the sequence.

Let the sequence, we have, be called current sequence.

Step 4: Let j be position of the first job with processing time = a in the current sequence. Let it be called pivot job.

Step 4.1: If the tardiness of the pivot job $\geq$ a, perform Step 4.1.1; otherwise perform Step 4.1.2.

Step 4.1.1:

a) Find the position k of a job with processing time = b which is encountered first as we move towards the left from the position of pivot job.

b) Construct an auxiliary sequence in which:

(i) first $(k-1)$ jobs and last $(n-j+1)$ jobs are same as in the current sequence.

(ii) The jobs occupying $(k+1)^{st}$, $(k+2)$, ... j^{th} positions in the current sequence are placed in k, $k+1$, ... $(j-1)^{th}$ positions.

(iii) The job occupying K^{th} position in current position is moved to j^{th} position.

(iv) If the tardiness of the auxiliary sequence $<$ tardiness of the current sequence, then the auxiliary sequence becomes the current sequence. Otherwise go to Step 4.1.2.

(v) Repeat Step 4.1 with the same pivot job until we have no job with processing time = b that precedes the pivot job.

Step 4.1.2: Select a job with processing time = a which is encountered first as we move to the right from the pivot job in the current sequence. Designate job in position j as new pivot job and go to Step 4.1. If no such job is found, terminate.

3.4 Proof of the Algorithm:

Consider a sequence A obtained by the algorithm.

(i) Take any arbitrary job j with processing time a the sequence A.Assume that it is moved to the left. So a job with processing time b that is to the left of job j will be moved to the right. Because of this, tardiness of the new schedule will increase. Otherwise we would have moved the job j in Step 4. So we cannot move job j to the left in sequence A. Suppose there are no jobs with processing time b that are preceding job j in sequence A, we cannot move job j to the left because all jobs with processing time a are to be in Edd. So any job j with processing time a cannot be moved to the left in sequence A.

(ii) Take any arbitrary job j with processing time a in the sequence A, is moved to right and a job with processing time b that is succeeding job j is moved to left.

The above job with processing time b might not have considered with the job with processing time a or it might have moved.

If the job with processing time b is moved to right while doing the algorithm means there is an improvement in tardiness of the schedule. Otherwise we would not have moved. So the job j with processing time a cannot be moved to right to decrease the tardiness of the schedule.

If the job with processing time b is not considered, the jobs with processing time a are sequenced by intervals

assignment initially. These jobs will have maximum earliness (b-a) or otherwise they become tardy.

In the algorithm, we are shifting the jobs with processing time a to the left to reduce the tardiness of the schedule. So we are not considering with processing time b that are at its right. This is because if the job with processing time b in position 'L' is moved to the left to position K (L $>$ K), and all the jobs with processing time a are moved from K to L-1 position to K+1 to L positions. The maximum decrease in tardiness of the job with processing time b is (L-K-1)a. The minimum increase in tardiness will be (L-K-1)a, since all jobs may be having earliness = (b-a). So the new schedule constructed will have $\geq$ the tardiness of initial sequence. So the jobs having processing time a are not moved to the right. If a job with processing time b, which occurred to the right of the job with processing time a in the initial sequence, is moved to the left, then the schedule constructed will definitely have more tardiness.

So any job with processing time a is moved to the right of its position in sequence A, the new schedule so constructed will not decrease the tardiness of the schedule.

From the above reasons, now we can say that the algorithm is giving an optimal solution.

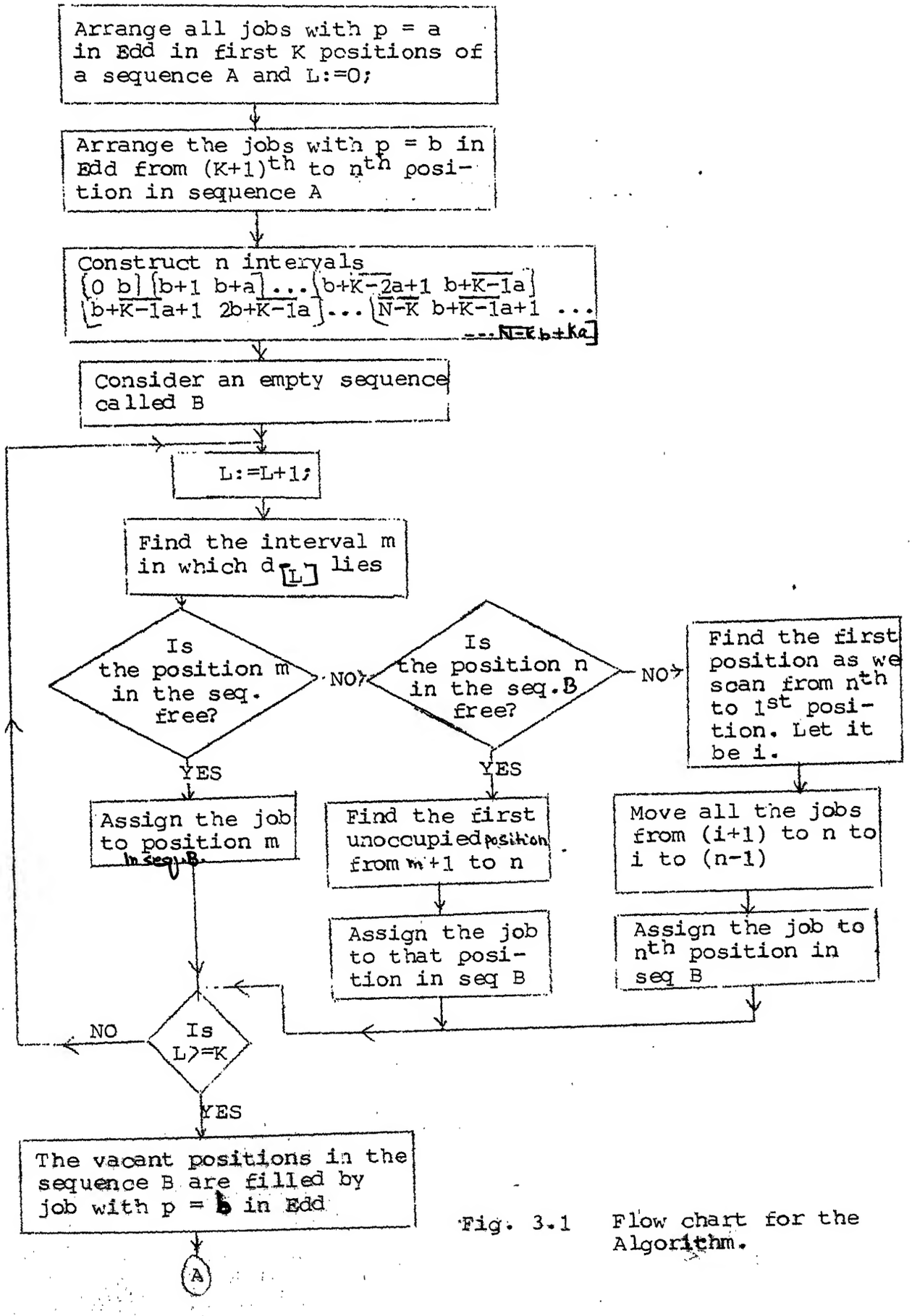

Fig. 3.1 Flow chart for the Algorithm.

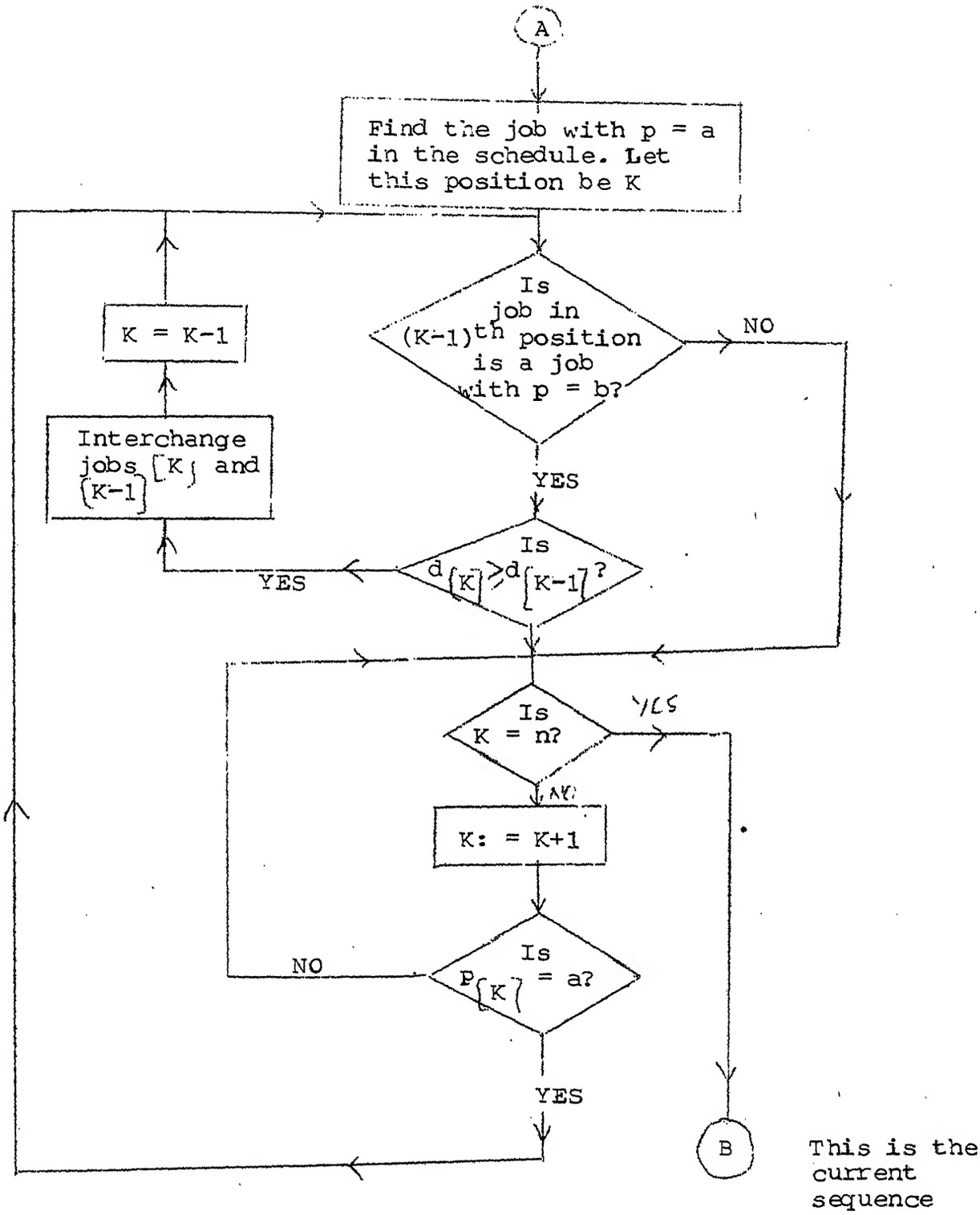

Fig. 3.1 (continues)

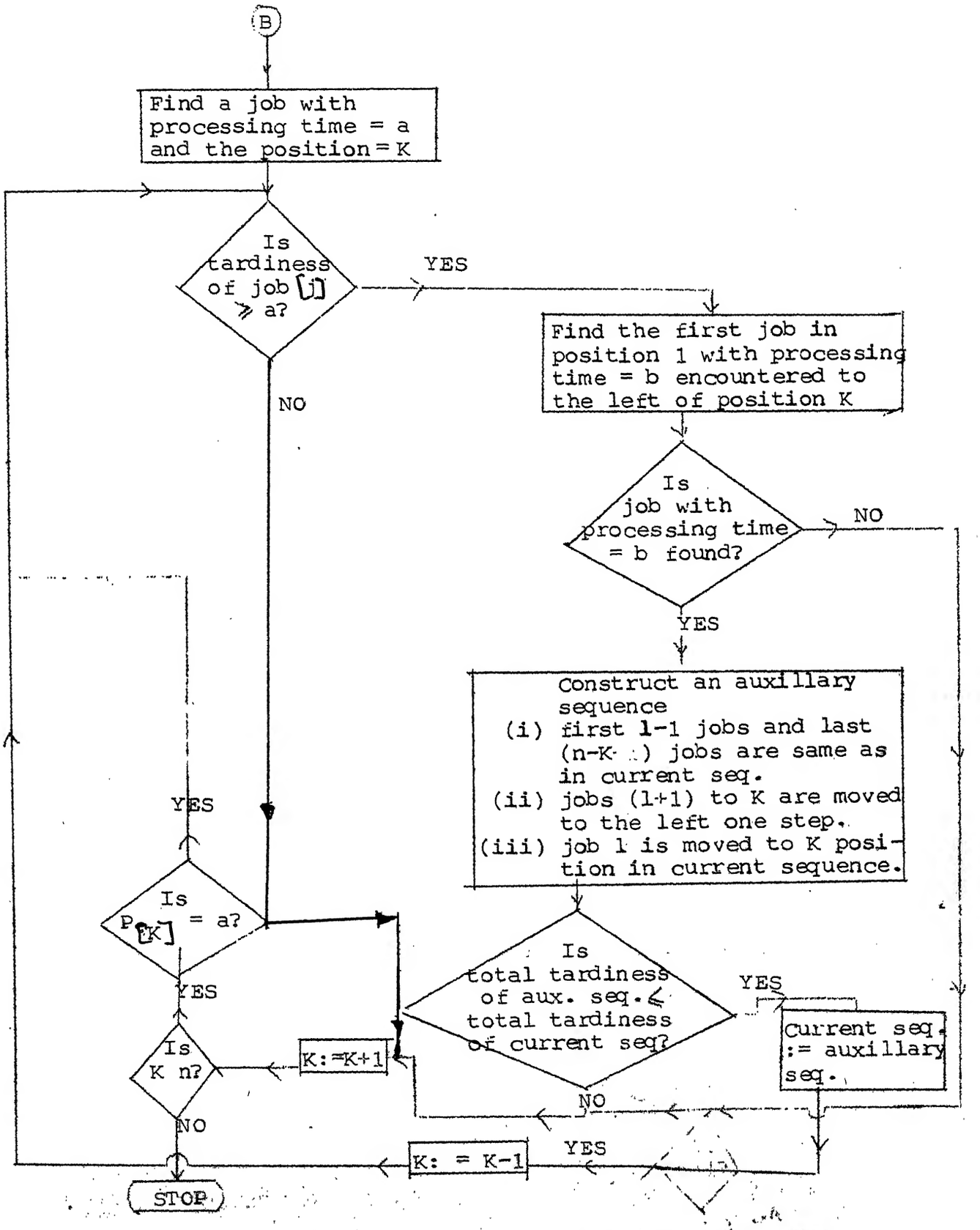

Fig. 3.1 (continues)

3.5 Complexity Analysis of the Algorithm:

One application of the following 4 steps are needed for solution. Let x be the no. of jobs with processing time = a.

Step 1: Keeping all the jobs with processing times a's in Edd and then all b's in Edd.
In worst case, all the jobs with processing time 'a' that are placed in Edd are having higher due dates than that of the job going to be placed. So at each step, the jobs that are placed have to be moved a step forward and then the new job can be placed. Similarly, job with processing time 'b' have to be placed in Edd. This may require on average and in worst case O(n log n) effort.

Step 2: In the worst case, for fixing the job with processing time 'a' is $O(x^2)$. This is because, for a job to be placed in the last position, all jobs may have to shift to left or it has to check the actual position and all positions to the right of its actual position. The other jobs with processing time 'b' can be placed in the remaining positions. This is of O(n-x).
Therefore total complexity of this step is $O(n^2)$ since $x \leq n$.

Step 3: In the worst case, all b's at starting may have higher due dates than all the a's. So each 'a' has to move (n-x) positions.

Therefore complexity of this step is $O(n^2)$.

Step 4: This consists of moving the job with processing time 'a' to left. In the worst case, the job with processing time 'a' that has not satisfied the condition, may have to move 'n-x' positions. So the complexity is $O(n-x)$. This has to be done for all a's. Therefore the complexity of this step is $O(n^2)$. Therefore the complexity of the algorithm is polynomial in nature.

3.6 Heuristics:

The ideas generated in the algorithm are extended to a general case of scheduling problem that resulted in heuristic procedures.

3.6.1 Heuristic-1:

Step 1: Arrange all the jobs in SPT order and if the processing time of some jobs are identical, then theyare arranged in EDD order. Let this be in a list of unscheduled jobs.

Step 2: a) Take the first job from the list of unscheduled jobs. Let it be the pivot job.

b) The condition to be satisfied to fix a pivot job into a position is that completion time of the pivot job $\geq$ the due date of pivot job.

c) Then position of the pivot job in the vacant position (from starting). The vacant positions prior

to the position of pivot job are assumed to be filled by the remaining unscheduled jobs (i.e., excluding pivot job) from the list in SPT order. If the condition in (b) is satisfied, go to Step (e). Otherwise repeat Step c until all the vacant positions are over.

d) If the pivot job cannot be positioned satisfying the condition in (b), then it is to be kept in the last vacant position that is available.

e) Once the pivot job is scheduled, it is removed from the unscheduled list and Step 2 is repeated until all jobs are positioned.

Step 3: Let i and j be such that the $p_{[i]} \geq p_{(j)}$ and $d_{(i)} > d_{(j)}$ and $i < j$ then jobs in position i and j are interchanged. Step 3 is repeated until no such change are possible. Let this be known as the current sequence.

Step 4:

Step 4.1: Find the first job in position from the starting of the current sequence whose completion time is more than its due date. Let this be pivot job j. Let $l = j$.

Step 4.2: Scan the current sequence from position (l-1) towards left and find a job in position K satisfying the condition:

$p_{[K]} > p_{\text{pivot job}}$

or

$p_{[K]} = p_{\text{pivot job}}$ and $d_{[K]} > d_{\text{pivot job}}$ and lateness $_{[K]} < 0$.

If no such job is found go to Step 4.5.

Step 4.3: Construct an auxiliary sequence in which

(i) all jobs upto $(K-1)^{th}$ position are same as in the current sequence.

(ii) last (n-j+1) jobs are same as in the current sequence.

(iii) all jobs from (K+1) to j in current sequence are moved to K to (j-1) positions.

(iv) job K is moved to j^{th} position.

Step 4.4: If the tardiness of auxiliary sequence is tardiness of current sequence then the auxiliary sequence will now become the current sequence and J:=J-1; l:=J; otherwise, l:=K. Repeat the Step 4.2 until all jobs preceding pivot job are over (i.e. l = 1).

Step 4.5: Repeat from Step 4.2, with a job which is right to the pivot job in the sequence. This job will now become the new pivot job, until all the jobs are over l = j.

The flow chart for the heuristic-1 is given in Figure 3.2.

3.6.2 Heuristic-2:

Step 1: Arrange all the jobs in EDD order and if the due dates of some jobs are identical, then they are arranged in SPT order.

Step 2:

Step 2.1: Find the first job in position j from the starting of the current sequence whose completion time

is more than its due date. Let this be pivot job.

Let 1 = j.

Step 2.2: Scan the current sequence from position (1-1) towards left and find a job in position K satisfying the condition:

$$p_{[K]} > p_{\text{pivot job}}$$

or

$$p_{[K]} < p_{\text{pivot job}} \text{ and } d_{[K]} > d_{\text{pivot job}} \text{ and lateness}_{[K]} < 0 \quad (1)$$

If no such job is found go to Step 2.5.

Step 2.3: Construct an auxiliary sequence in which

(i) all jobs upto $(K-1)^{th}$ position are same as in the current sequence.

(ii) last (n-j+1) jobs are same as in the current sequence.

(iii) all jobs from (K+1) to j in current sequence are moved to K to (j-1) positions.

(iv) job K is moved to j^{th} position.

Step 2.4: If the tardiness of auxiliary sequence is tardiness of current sequence then the auxiliary sequence will now become the current sequence and j:=j-1; 1:=J; otherwise, 1:=K. Repeat the Step 2.2 until all jobs preceding pivot job are over (i.e. 1 = 1).

Step 2.5: Repeat from Step 2.2, with a job which is right to the pivot job in the sequence. This job will now become the new pivot, until all the jobs are over i = j.

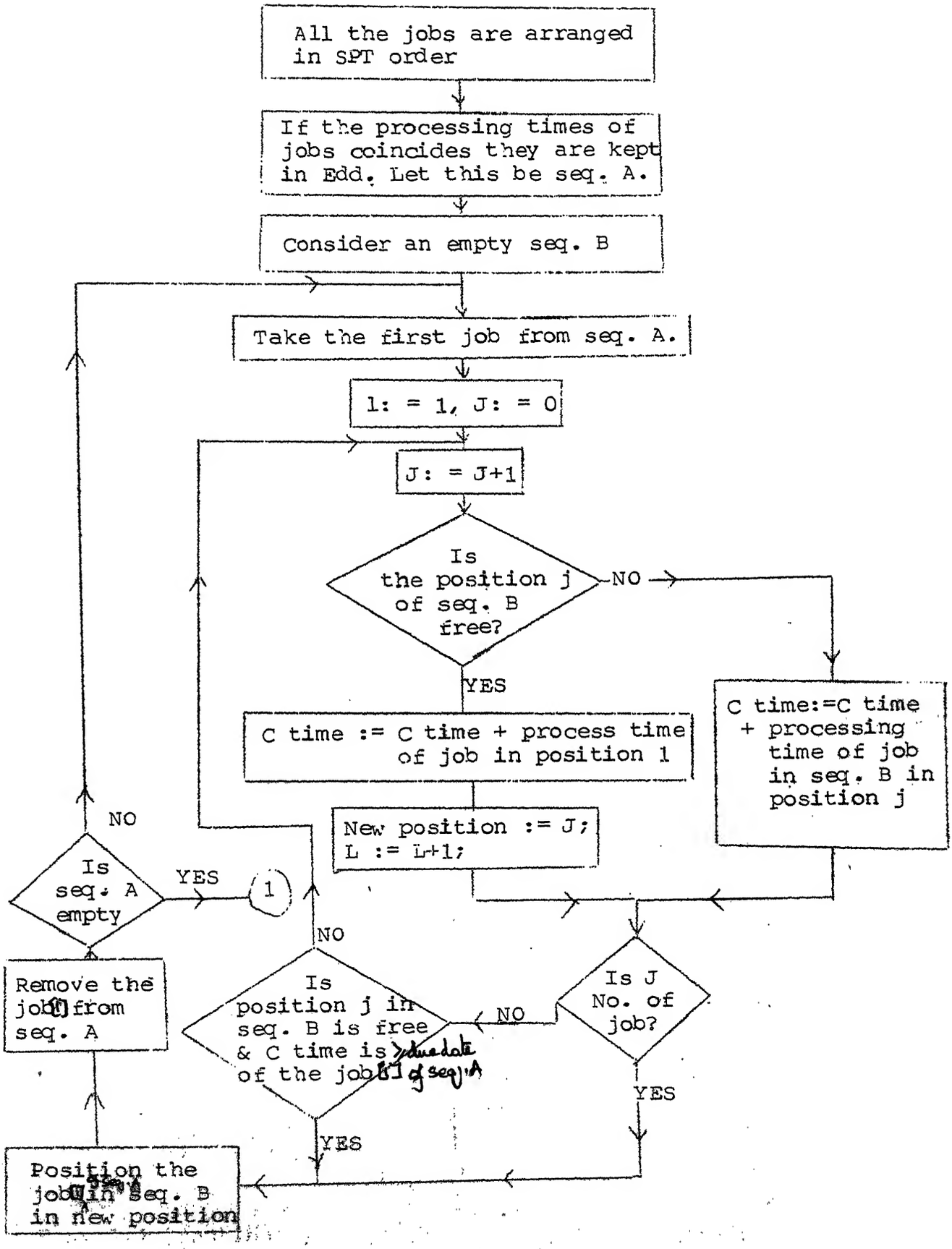

Fig. 3.2 Flow Chart for Heuristic 1.

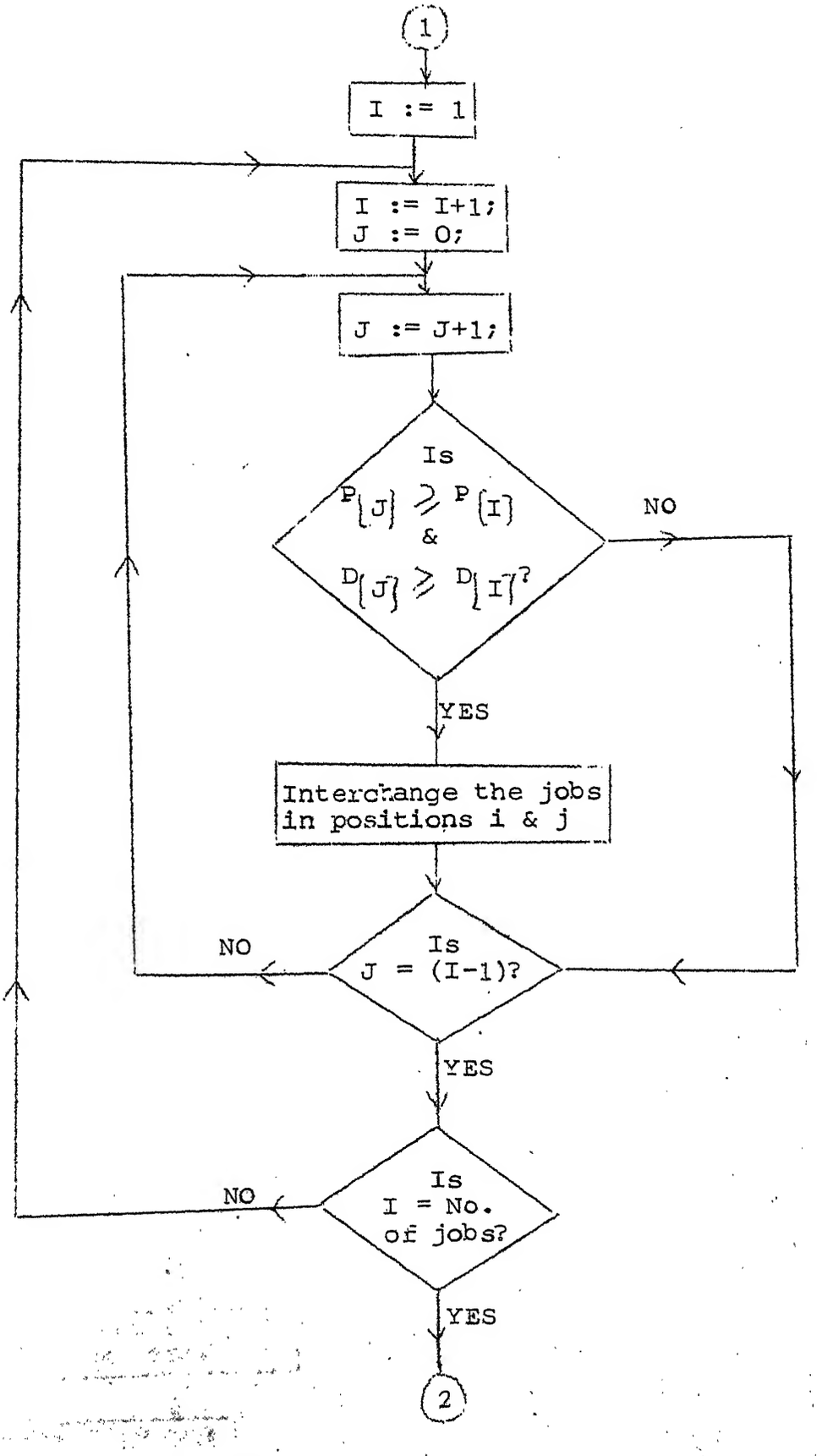

Fig. 3.2 (continues)

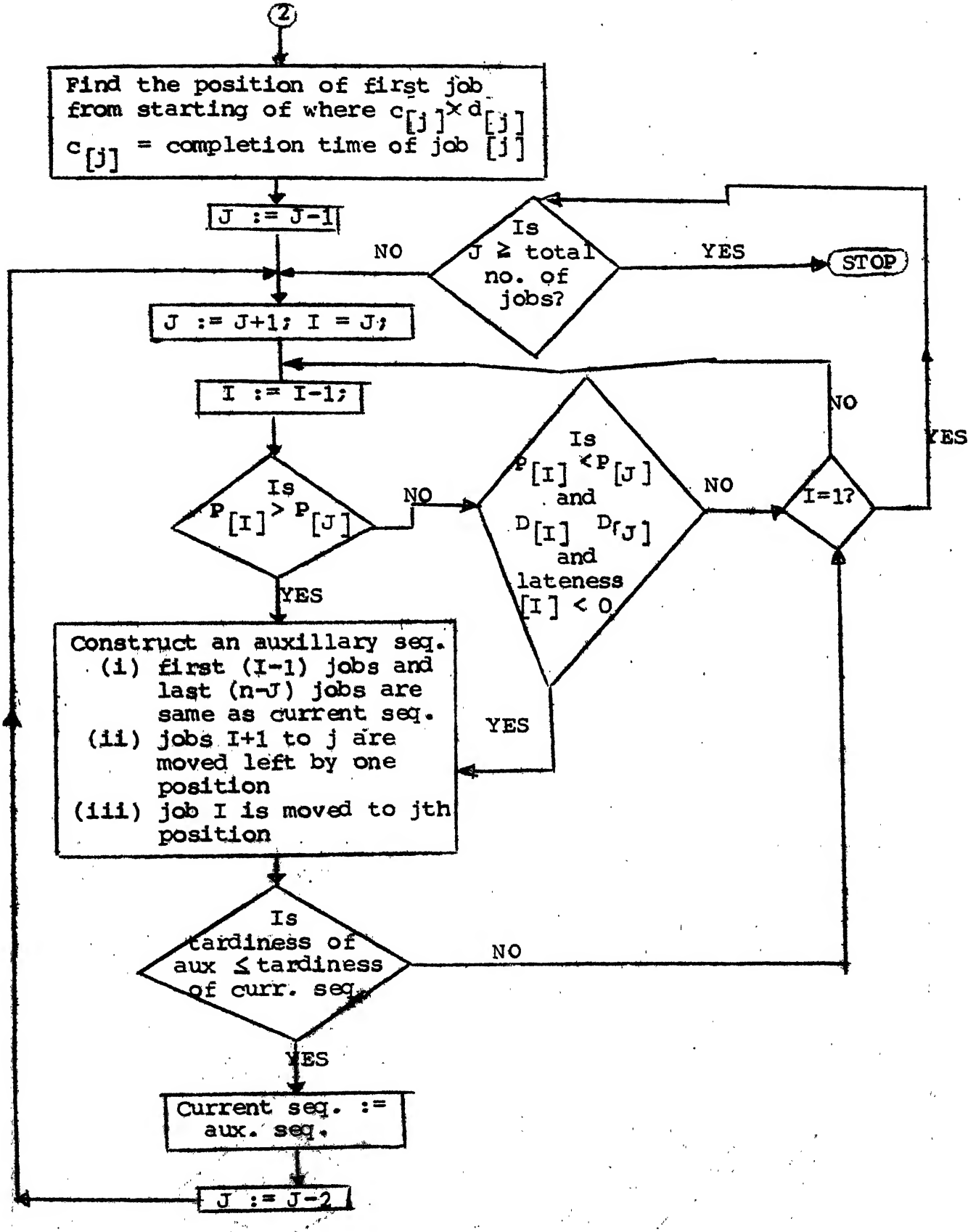

Fig. 3.2 (continues)

Chapter IV

COMPUTATIONAL STUDY

4.1 Performance of the Algorithm:

A computational study was undertaken where the performance of the algorithm has been compared with the Dynamic Programming approach. The problems considered in the study had the following characteristics.

The processing times of the job are taking only two values 1 and 2. Let P denote the sum of the processing times of all the jobs, the due dates are generated such that $d \sim U[1, P]$. The number of jobs considered are between 3 and 9.

The results of the study are presented in Table 4.1, 4.2. As can be readily observed, the performance of the algorithm is considerably better compared to the solution using DP approach. Moreover the algorithm shows increasingly better performance with the increase in problem size.

4.2 Performance of the Heuristic:

The second study considered was to compare the heuristic procedures developed with Wilkerson-Irwin algorithm [PB-14]. Test problems of various sizes ranging from n = 20 to n = 100 are randomly generated using a methodology suggested by Fisher [8].

Let τ be tardiness factor of a set of jobs

R be due date range of jobs.

Tardiness factor of a set of jobs means the average number of jobs that would be late if the jobs are randomly scheduled.

The due date range of jobs refers to the range of distribution from which the due dates of jobs are drawn.

The method used to generate the data for any problem is as follows. We first generate the processing times p_i for each of the jobs such that $p_i \sim U\ [50 - 150]$. Then if P is sum of the processing times generated, the due dates d_i are generated such that $d_i \sim U\ [P(1 - \tau - R/2)\ P(1 - \tau + R/2)]]$.

In this study, the total tardiness, the number of tardy jobs of the schedule obtained by the two heuristic procedures and Wilkerson-Irwin algorithm are compared. The values τ and R that are considered are from 0.2 to 0.8 (both). The results of the study are presented in Tables 4.3, 4.4 and 4.5.

Initially the heuristics were compared with Wilkerson-Irwin algorithm for a 50 job problem and processing time range is between 50-150; three different sets of problems were generated for a particular value of τ and R and the average was taken. From the results obtained, it can be observed that Heuristic 1 is giving better results except when τ and R are equal. In most of the cases, the number of tardy jobs that are given by Heuristic 1 is less than by the other two. Heuristic 2 invariably yields better solutions for all the cases considered. The number of tardy jobs

obtained is the schedule is less than Wilkerson-Irwin algorithm in most of the cases. However the CPU time taken by heuristics was higher than Wilkerson-Irwin algorithm.

TABLE 4.1 COMPARATIVE PERFORMANCE OF OUR ALGORITHM (for 100 problems)

	no. of Jobs	Dynamic Programming Approach. CPU secs.	Our Algorithm CPU (secs).	$n = \frac{\text{Time taken for DP}}{\text{Time taken by our procedure}}$
I.	3 Job			
	1 1 2	0.15	0.123	1.2
	1 2 2	0.147	0.114	1.28
II.	4 Job			
	1 2 1 1	0.316	0.161	1.96
	1 2 1 2	0.320	0.159	2.00
	1 2 2 2	0.315	0.155	2.03
III.	5 Job			
	1 1 2 1 1	0.70	0.228	3.07
	1 2 2 1 1	0.694	0.222	3.12
	1 2 2 1 2	0.691	0.194	3.56
	1 2 2 2 2	0.689	0.176	3.91
IV.	6 Job			
	111211	1.568	0.278	5.64
	121211	1.568	0.263	5.96
	122211	1.571	0.260	6.04
	122212	1.562	0.240	6.50
	122222	1.562	0.225	6.94
V.	7 Job			
	1112111	3.525	0.34	10.36
	1112121	3.52	0.333	10.57
	1122121	3.51	0.327	10.73
	1222121	3.514	0.307	11.44
	1222221	3.506	0.282	12.43
	1222222	3.492	2.54	13.74
VI.	8 Job			
	11121111	7.923	0.416	19.04
	11221111	7.921	0.403	19.68
	12221111	7.924	0.390	20.32
	12221211	7.901	0.383	20.62
	12221221	7.867	0.363	21.67
	12222221	7.878	0.332	23.72
	12222222	7.859	0.296	26.55

TABLE 4.1 (continued)

$$n = \frac{\text{Time taken for DP}}{\text{Time taken by our procedure}}$$

VII. 9 Job

111211111	17.66	0.504	35.03
111211211	17.565	0.491	35.77
111212211	17.486	0.479	36.50
112212211	17.493	0.468	37.38
112212221	17.449	0.444	39.29
112222221	17.445	0.400	43.61
122222221	17.537	0.384	45.67
122222222	17.519	0.345	50.77

TABLE 4.2 SUMMARY OF THE TABLE 4.1

Jobs	$n = \frac{\text{Time taken for DP}}{\text{Time taken by our procedure}}$
3	1.24
4	1.99
5	3.4
6	6.2
7	11.54
8	21.65
9	40.5

TABLE 4.3 COMPARATIVE PERFORMANCE OF THE HEURISTICS (varying the parameters)

Processing time range 50-150; No. of jobs = 50; (No. of problems for a value of τ & R = 3)

		Wilkerson procedure		Heuristic 1			Heuristic 2			% change in total tardiness between Wilkerson & Heuristic 1	% change in total tardiness between Wilkerson & Heuristic 2
τ	R	Total tardiness	CPU Secs	Total tardiness	CPU Secs	No. of times heuristic performed better	Total tardiness	CPU secs	No. of times neuristic 2 performed better		
0.2	0.2	1562	0.025	1688	0.15	1	1544	0.71	3	-8.0	1.16
0.4	0.2	11718	0.114	11510	0.84	3	11475	1.48	3	1.77	2.07
0.6	0.2	32200	0.246	31568	1.4	3	31396	2.43	3	1.96	2.50
0.8	0.2	62816	0.378	62094	1.23	3	61728	2.70	3	1.15	1.70
0.2	0.4	12	0.026	12	0.21	3	12	0.51	3	0	0
0.4	0.4	7087	0.084	7333	0.78	1	7049	1.1	3	-3.47	0.53
0.6	0.4	26953	0.223	26825	1.0	2	26657	2.1	3	0.47	1.09
0.8	0.4	59663	0.38	59648	0.3	3	59663	2.1	3	0.025	0
0.2	0.6	0	0.02	0	0.1	3	0	0.5	3	0	0
0.4	0.6	3227	0.044	3187	0.782	3	3158	0.85	3	1.24	2.10
0.6	0.6	22182	0.19	22484	0.88	1	21947	1.8	3	-1.36	1.05
0.2	0.8	0	0.019	0	0.1	3	0	0.51	3	0	0
0.4	0.8	400	0.22	400	0.53	3	400	0.51	3	0	0
0.6	0.8	19539	0.157	19535	0.78	2	19525	1.36	3	0.02	0.07
0.2	1.0	0	0.025	0	0.1	3	0	0.52	3	0	0
0.4	1.0	25	0.02	25	0.356	3	25	0.5	3	0	0

TABLE 4.4 DETAILED PERFORMANCE OF THE HEURISTICS (varying number of jobs)

Processing time range 50-150; $\tau = 0.8$; $R = 0.2$

No. of Jobs	Wilkerson			Heuristic 1			Heuristic 2		
	N_T	Total tardiness	CPU Secs	N_T	Total tardiness	CPU Secs	N_T	Total tardiness	CPU Secs
20	19	10745	0.017	15	10643	0.036	16	10565	0.117
30	25	23354	0.082	23	23334	0.21	20	23201	0.462
40	34	41250	0.22	27	40822	0.57	32	40893	2.42
50	43	63636	0.33	37	63280	0.83	38	63279	6.60
60	50	90173	0.743	43	89624	2.43	44	89405	6.60
70	60	121681	1.0	46	120319	3.50	51	120675	10.0
80	68	157312	1.63	51	154201	6.90	54	153695	19.5
90	79	200067	2.4	60	199619	11.0	69	198651	24.0
100	85	254819	3.0	62	252041	24.0	70	248695	51.0

(N_T = no. of Tardy Jobs obtained)

TABLE 4.5 SITUATION WHERE HEURISTIC 1 PERFORMS POORLY

Processing time range 50-150; $\tau = 0.4$; $R = 0.4$

No. of Jobs	Wilkerson			Heuristic 1			Heuristic 2		
	N_T	Total tardiness	CPU Secs	N_T	Total tardiness	CPU Secs	N_T	Total tardiness	CPU Secs
20	5	1219	0	5	1219	0.058	5	1219	0.048
30	8	3108	0.018		3122	0.199	8	3108	0.21
40	13	5820	0.06	11	5885	0.45	13	5820	0.70
50	16	7260	0.081	10	7140	0.74	10	7146	0.912
60	20	10825	0.134	18	10990	1.47	19	10820	2.60
70	18	12740	0.225	12	13195	2.25	18	12740	2.90
80	22	17328	0.28	20	17040	3.70	22	16627	5.90
90	27	23488	0.42	20	22975	5.40	22	22755	11.0
100	33	28545	0.59	20	28962	8.30	24	28203	13.0

(N_T = no of Tardy Jobs obtained)

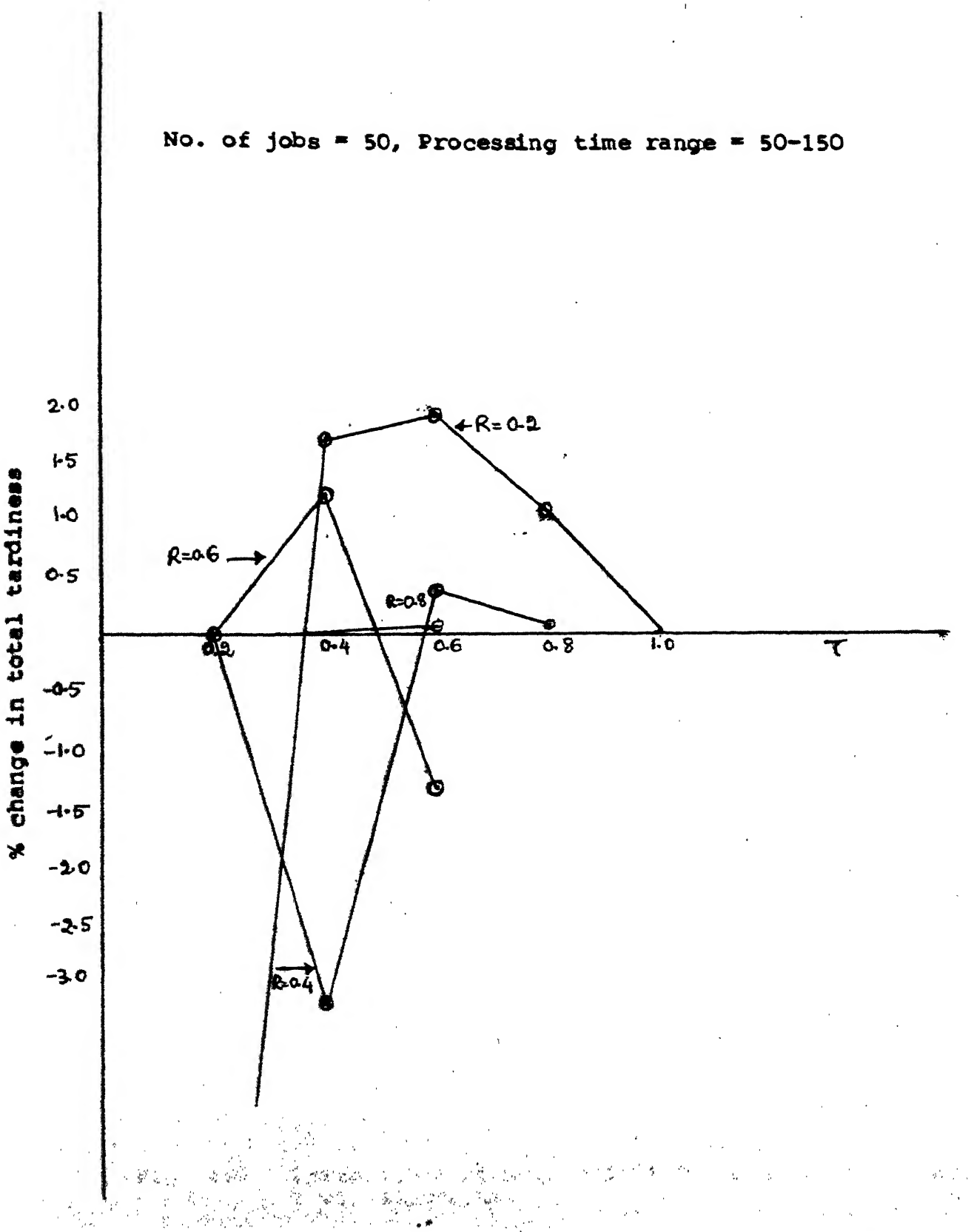

Fig. 4.1 Performance of Heuristic-1 Compared with Wilkerson-Irwin Algorithm.

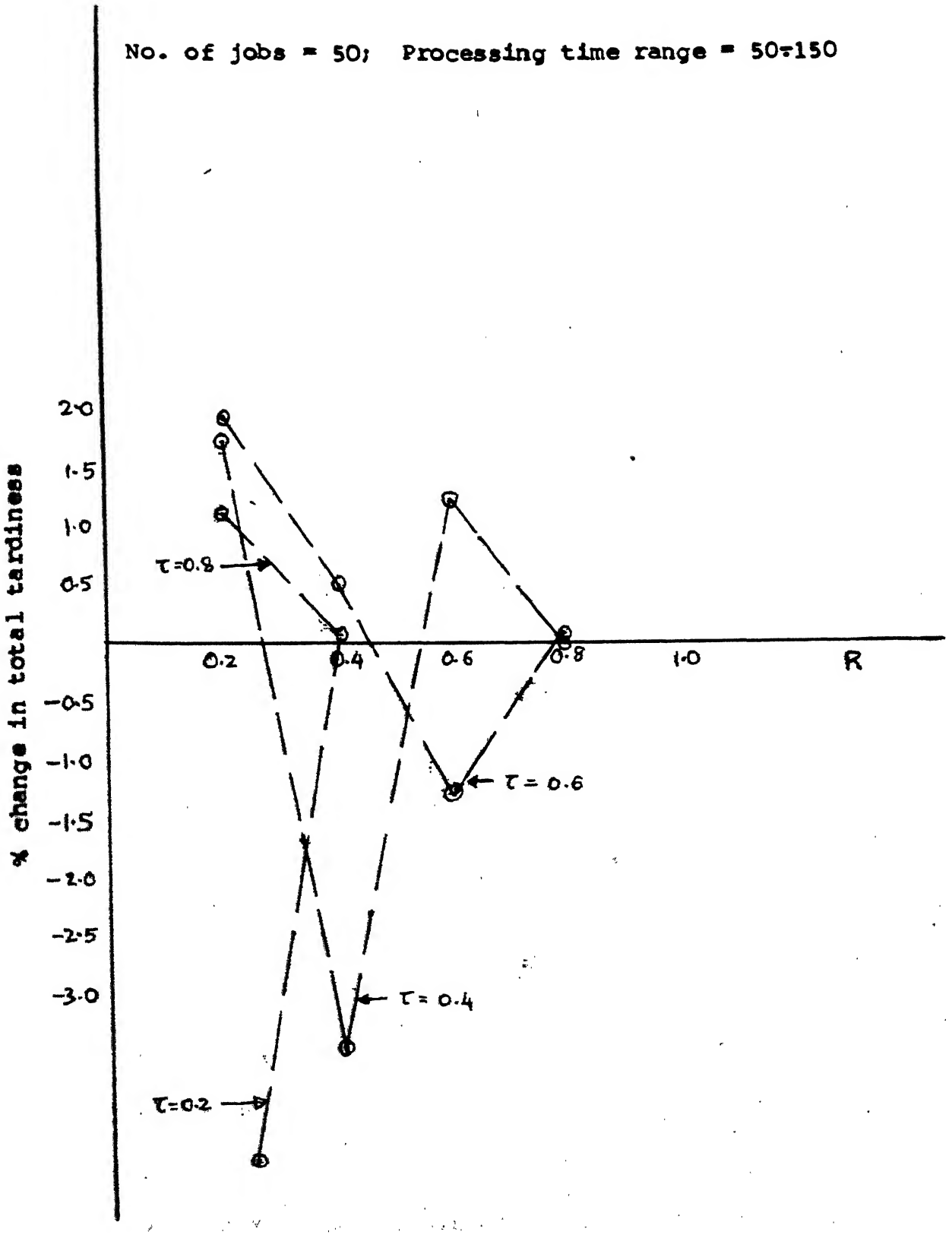

Fig. 4.2 Performance of Heuristic-1 Compared with Wilkerson-Irwin Algorithm.

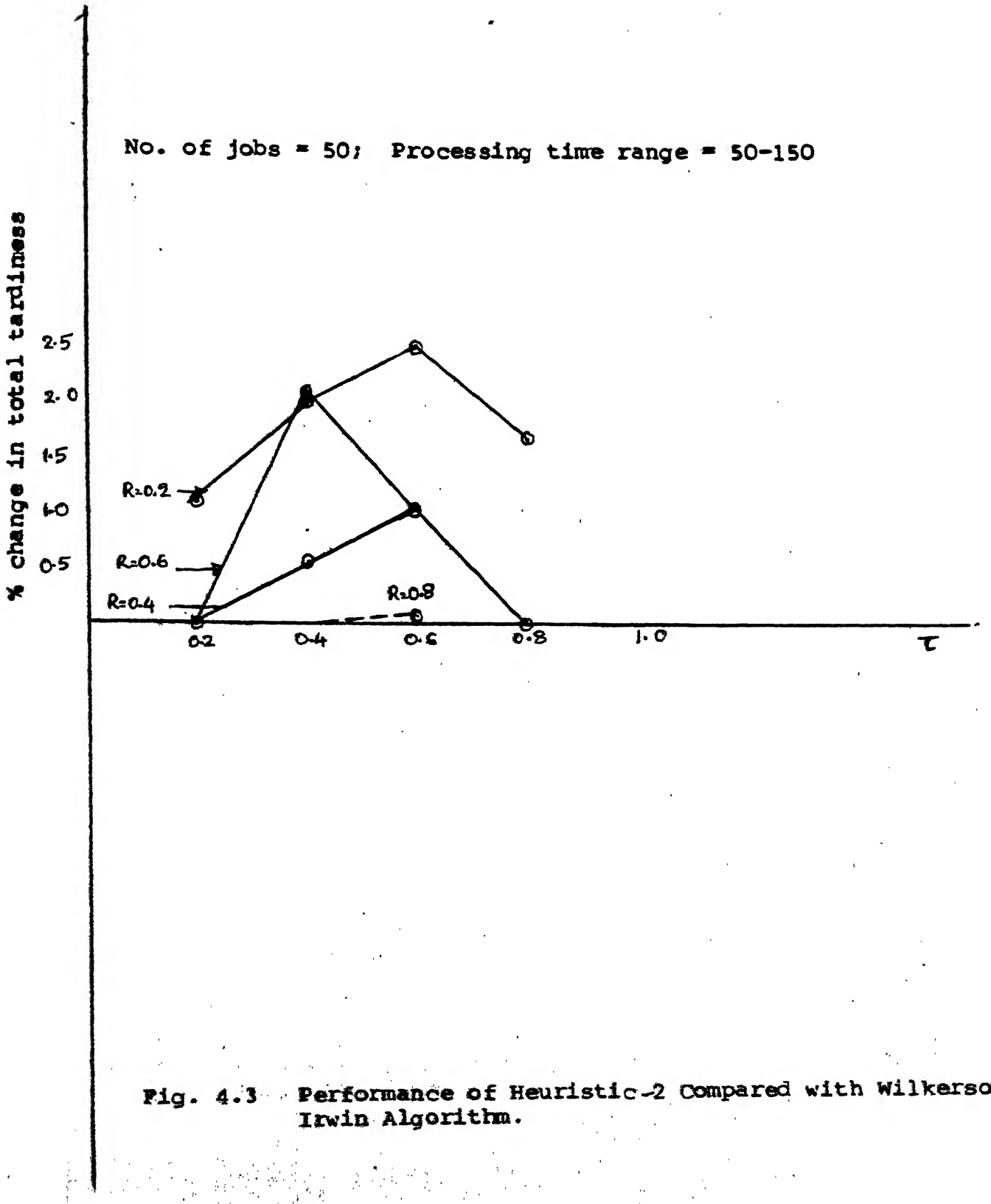

Fig. 4.3 Performance of Heuristic-2 Compared with Wilkerson-Irwin Algorithm.

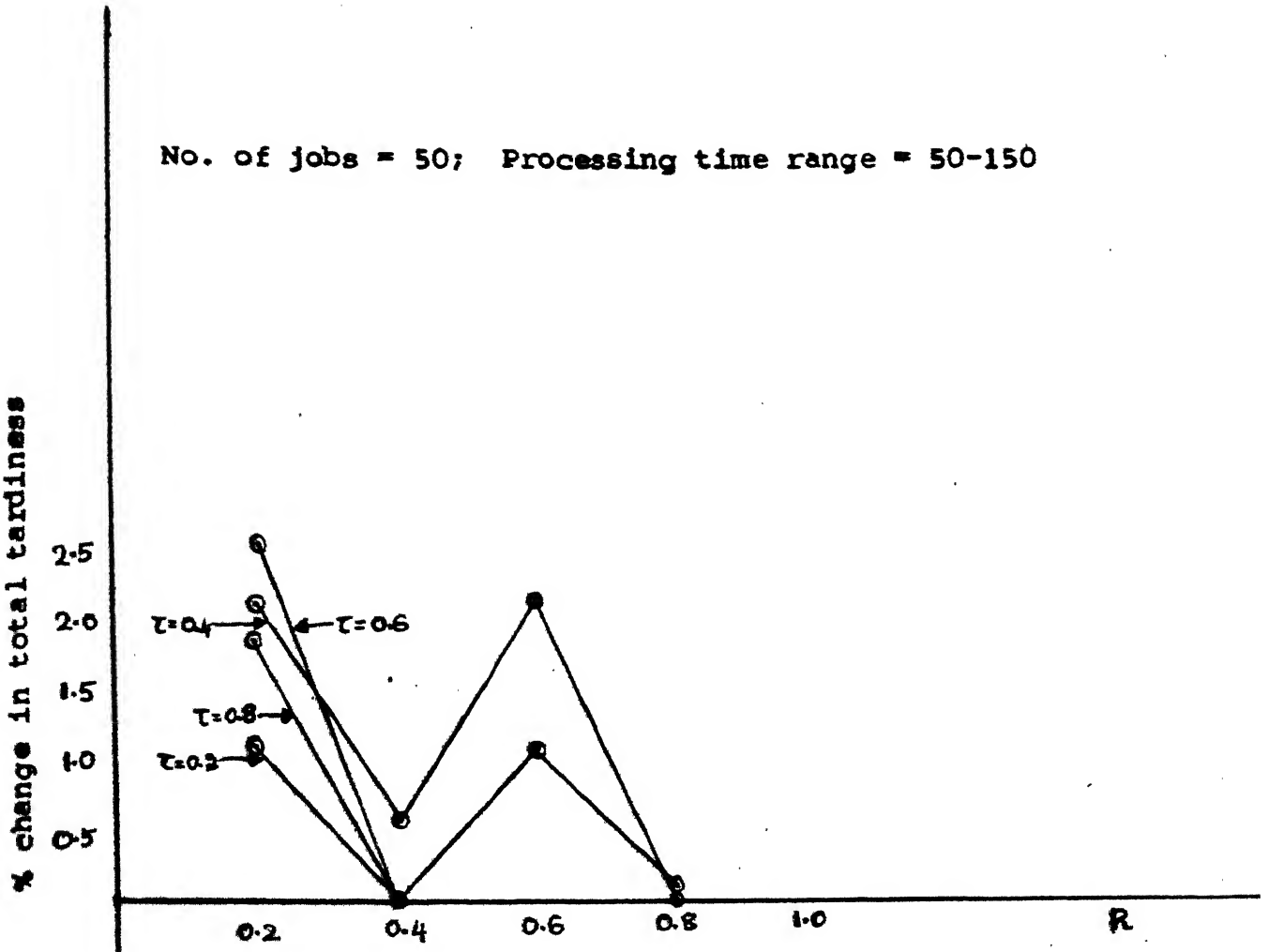

Fig. 4.4 Performance of Heuristic-2 Compared with Wilkerson-Irwin Algorithm.

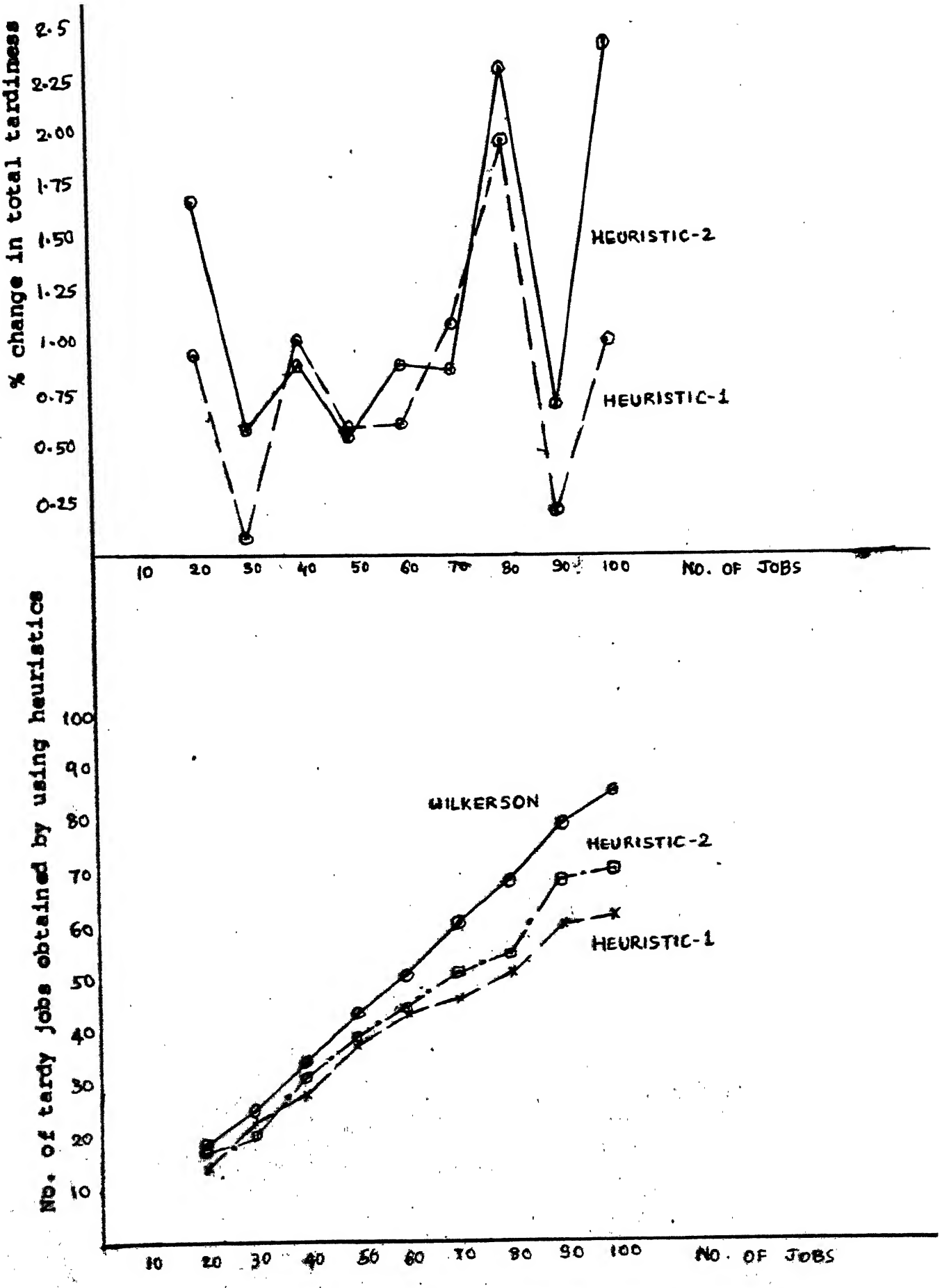

Fig. 4.5 Performance of Heuristics(Processing Time Range 50-150, $\tau = 0.8$, $R = 0.2$).

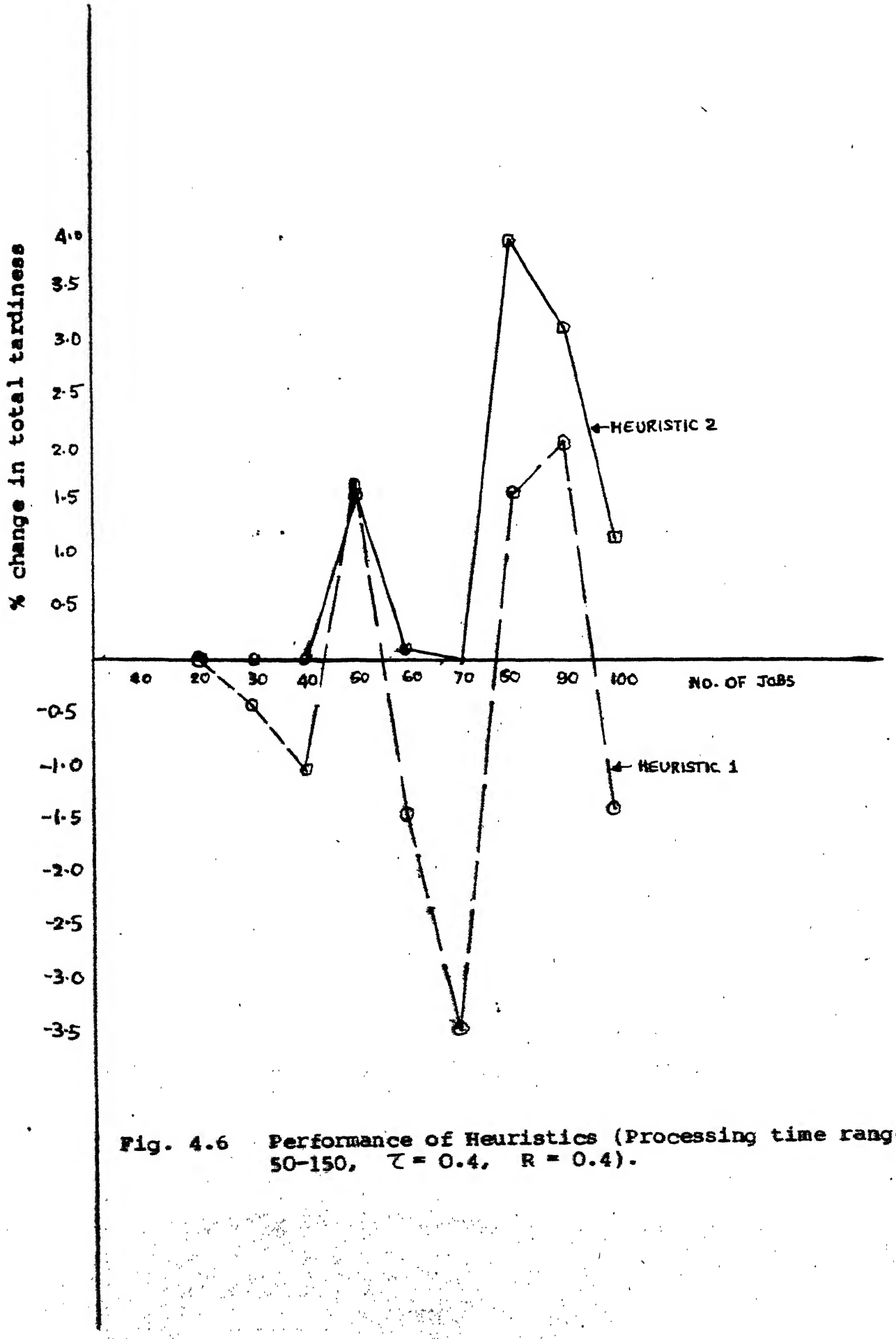

Fig. 4.6 Performance of Heuristics (Processing time range = 50-150, $\bar{\tau}$ = 0.4, R = 0.4).

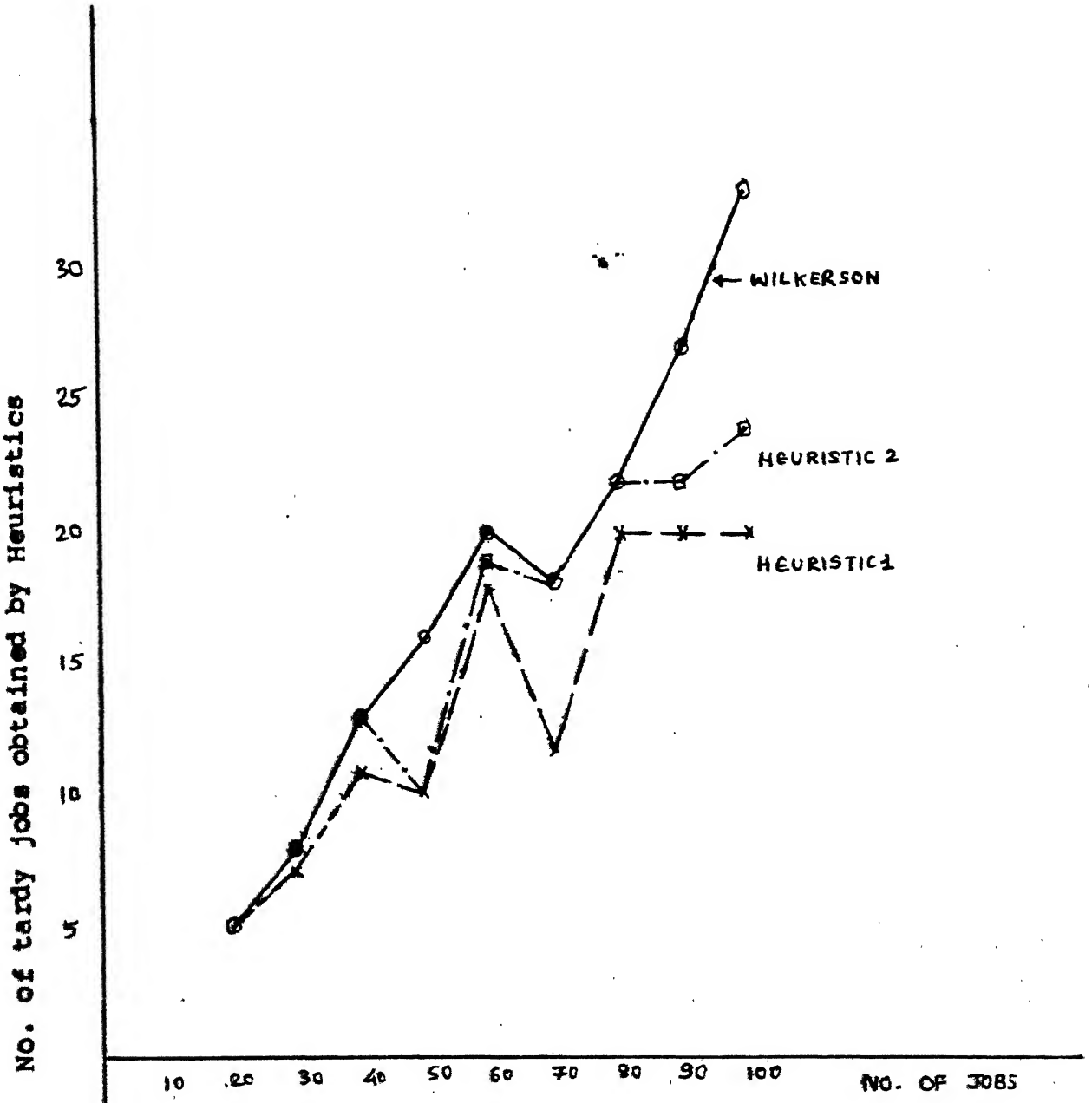

Fig. 4.7 Performance of Heuristics (Processing time range = 50-150, $\bar{T}$ = 0.4, R = 0.4).

Chapter V

CONCLUSIONS AND RECOMMENDATIONS

The main result of the thesis is the development of a polynomial time algorithm for a special case of single machine scheduling problem. Though polynomial time algorithms are available when the objective function is to minimise the mean flow time etc. no polynomial algorithm is available for minimisation of mean tardiness. In this thesis, we consider a restriction where the processing times of the different jobs are limited to two values only. With this limitation, we exploit the structure of the problem to get a polynomial time algorithm.

Computational studies to evaluate the average performance of the algorithm yield promising results. However the extension of the algorithm to more general case for example three distinct processing times appear to be difficult even though we studied these cases extensively. The only promise seems to be to imbed the ideas into heuristic procedures for the solution of general problem. This was done which resulted in two heuristic procedures developed in this thesis. One of them has an excellent performance when compared with the well-known Wilkerson-Irwin algorithm, the standard procedure known in the literature. The other heuristic has a mixed performance sometimes giving better results, sometimes poorer results compared to the Wilkerson-Irwin algorithm. Extensive

studies indicating various patterns of performance have been reported in the thesis.

Recommendations:

The special case of the single machine scheduling problem studied in this thesis is unlikely to be of any practical interest. However the insights gained from a study of such a restricted problem would indeed be valuable to study the general problem as such even though a straightforward extension appears to be impractical. Innovative approaches should be attempted in this direction. Alternately several special cases each admitting polynomial time algorithms may be identified which found the basis of totally different approach to the solution of the scheduling problem.

Considerable scope exists for the study of heuristic procedures based on the ideas developed in this thesis. This may ultimately lead to heuristics that guarantee epsilon optimum solutions for the general scheduling problem.

References

1. Baker K.R. (1974), Introduction to Sequencing and Scheduling, John Wiley and Sons, New York.

2. Emmons H. (1969), "One machine sequencing to minimise certain functions of job tardiness", Opns. Res., Vol. 17, 701-715.

3. Robert McNaughton (1959), "Scheduling with deadlines and loss functions", Mgmt. Sci., Vol. 6, No. 1, 1-12.

4. Elmaghraby S.E. (1968), "The one machine sequencing problem with delay costs", Jl. of Ind. Engg., Vol. 19, No. 2, 105-108.

5. Shwimer J. (1972), "On the N-job, one machine, sequence independent scheduling problem with tardiness penalties: A branch and bound solution", Mgmt. Sci., Vol. 18, No. 6, B301-B313.

6. Rinnooy Kan, A.H.G. et.al. (1975), "Minimising total costs in one machine scheduling", Opns. Res., Vol. 23, No. 5, 908-927.

7. Peng Siow and Morton, T.E. (1985), "An investigation of beam search for scheduling", Proceeding of TIMS/ORSA, Boston meeting, April 29-May 1, 1985.

8. Fisher, M.L. (1976), "A dual algorithm for one machine scheduling problem", Math. Prog., Vol. 11, 229-251.

9. Graves, S.C. (1981), "A review of production scheduling", Opns. Res., Vol. 29, 647-675.

10. Lawler E.L. (1964), "On scheduling problems with deferral costs", Mgmt. Sci., Vol. 11, No. 2, 280-288.

11. Potts, Wassenhove (1985), "A branch and bound algorithm for the total weighted tardiness problem", Opns. Res., Vol. 33, No. 2, 363-377.